单源最短路

2019年3月11日　　14:54

-
- 单源最短路径

一. 图和存储方式
   1. 无向边视作两条相反的有向边，以后都讨论有向图
   2. 二维数组法存储邻接矩阵：
      i. w[i][j]存储i到j的权值
      ii. i==j时存储0，i到j无路时存储正无穷
   3. 数组法存储邻接表：
      i. 长度n的数组fst存储n个点的最新输入的边的下标
      ii. 长度m的数组to和val存储边的终点和权
      iii. 长度m的数组nxt存储同起点的下条边的下标
      iv. tot存边的数量
      v. 空间复杂度O（n+m）

```
inline void add(int x,int y,int v=0){
	to[++tot] =y;
	val[tot] =v;
	nxt[tot] =fst[x];
	fst[x] =tot;
}
```

      vi. //遍历x起点的所有边

```
for(int i=fst[x]; i; i=nxt[i]){  //遍历x起点的各终点
	int y=to[i], z=val[i]; }
```

二. 单源最短路径SSSP
   1. Single Source Shortest Path：固定起点到其他点的最小权路径
      i. 点V的数量=n，边E的数量=m
      ii. 数组dist[i]表示从起点1到终点i的最短路径长度
   2. relaxation松弛操作：对权值的上界做估计
      i. 满足“边权三角形不等式”时即可对y松弛：dist[y]<=dist[x]+z
   3. Dijkstra算法O（n方）
      i. 适用于无负权值的图
      ii. 初值：dist[1]=0，其他为正无穷大；v数组全0（访问标记）
      iii. 思路：找未访问过的、dist[x]最小的节点x，遍历所有从它出发的点的终点，转移dist[y]=min(dist[y],dist[x]+w[x][y])
      iv. 注：未访问过的、dist最小的x已经确认不可能有更短路了（无负边的情况下），所以可以用它去尝试更新其他点

```
void dijkstra(){
	memset(d,0x3f,sizeof(d));          //初值正无穷
	memset(v,0,sizeof(v));             //初值未访问
	d[1]=0;
	for_(i,1,n){        //遍历n-1个点
```

```
int x=0; //权最小点的序号
for__(j,1,n)
    if(!v[j] && d[j]<d[x])
        x=j;
v[x]=1;         //及时标记
for__(y,1,n)    //更新所有点
    d[y]=min(d[y],d[x]+w[x][y]); }}
```

4. 二叉小根堆（优先队列）和邻接表实现Dijkstra算法//顺便找最小环
    i. 获取最小值的复杂度O（logn），执行扩展的复杂度O（logn），整体复杂度约O（(m+n)logn）
    ii. 初值：堆里可以只存<0,1>，因为第一次while循环所有边就会自动更新所有其他点进堆

```
int n,m,tot=1;      //点数，边数，边号
int fst[MN],to[MN<<1],nxt[MN<<1],val[MN<<1];

inline void add(int x,int y,int v=0){
    to[++tot] =y;
    val[tot] =v;
    nxt[tot] =fst[x];
    fst[x] =tot;
}
bool v[MN];
ll d[MN];     //到源点的距离
priority_queue<pair<ll,int> >q;
//二维是节点编号，一维是dist相反数（直接实现小根堆）
void dij(int src){
    ms(d,0x3f), ms(v,0);
    int y,z;
//    d[src]=0;    q.push(make_pair(0,src));
    for(int i= fst[src]; i; i= nxt[i]){
        int y= to[i], z= val[i];
        d[y]= z;
        q.push(make_pair(-d[y],y));
    }
    while(q.size()){
        int x= q.top().second; q.pop();
        if(v[x]) continue;
        v[x]=1;
        for(int i= fst[x]; i; i= nxt[i]){
            y= to[i], z= val[i];
            if(d[y]> d[x]+z){
                d[y]= d[x]+z;
                q.push(make_pair(-d[y],y));
            }
        }
    }
}
```

5. 队列优化Bellman-Ford算法即SPFA（Shortest Path Fast Algorithm）
    i. Bellman-Ford：扫描每条边，满足三角形不等式的话进行松弛操作，n次扫描后，如果还可以松弛，说明有负圈
    ii. 队列优化：扫描队头x点为起点的所有出边，若满足三角形不等式，且终点y不在队列，则插进队尾（相当于可能重复入队的广搜松弛）
    iii. 最糟复杂度O（nm）
    iv. //顺便判负环

```
int n,m,tot,dst[MN],fst[MN];      //点数、边数、边号、最短路
```

```
int to[MN<<1],val[MN<<1],nxt[MN<<1];
inline void add(int x,int y,int v=0){
    to[++tot] =y;
    val[tot] =v;
    nxt[tot] =fst[x];
    fst[x] =tot;
}
int inq[MN];
queue<int> q;

bool spfa(int src){ //src为源点最短路，返回是否有环
    ms(dst,0x3f); dst[src]=0;
    ms(inq,0); inq[src]=1;
    while(q.size()) q.pop(); q.push(src);  //起点
//    int cnt=0; q.push(cnt);
    while(!q.empty()){
        int x=q.front(); q.pop();
//          cnt=q.front(); q.pop();
//          if(cnt>n) return 1; //有负环
        inq[x]=0;    //x已不在队列中
        for(int i=fst[x]; i; i=nxt[i]){  //遍历x起点的各终点
            int y=to[i], z=val[i];
            if(dst[y]>dst[x]+z){
                dst[y]=dst[x]+z;
                if(!inq[y])  //y不在队列中
                    q.push(y),
//              q.push(cnt+1),
                    inq[y]=1; }}}
    return 0;//没负环
}
```

三. 应用

1. Dijkstra实现找**最小环**
   i. 设非源点结点y的dist初值为w[源点][y]，而设源点的dist为无穷大，且不令vis[源点]=0，就能从dist[y]+w[y][源点]更新出环，存储在dist[源点]
   ii. 相当于用源点更新好其他结点后，再给源点的dist赋为无穷，并令vis为0
2. Dijkstra实现“伪多源”最短路径
   i. 即n个源的最短路径中的最短值，其实是不用跑n次的
   ii. 新建一个结点，让它连向n个源，规定这n条边的权为0
   iii. 则以该结点为源点的单源最短路即为所求
3. SPFA实现差分约束系统（满足多个形如a[y]-a[x]<=z的条件的a数组）
   i. 变形得到形如边权三角形不等式a[y]<=a[x]+z的式子，因此可通过连有向边(x,y)=z再求最短路实现求解
   ii. 有负环时无解，因此一般用spfa求最短路
   iii. 如果条件形如a[y]-a[x]>=z，既可用y到x反向连边求最长路，正环无解，也可两边同乘负一，跑最短路，负环无解
      - 例题

四. 构造X，求关于矩阵C的最小ΣΣ Cij * Xij（USSTD2G）

1. 对X要求：第1行除了第1列的总和为1，第n列除了第n行的总和为1，对[2,n-1]的每个i，要求i行总和=i列总和
2. 引：将矩阵X乘上C，将C视作邻接矩阵，发现即为要求起点1出度1，终点n入度

1，其他点出入度相同

3. 引：dij跑1~n最短路显然符合上述条件（其他点的出入度都为0或都为1）
4. 转：还有一个隐藏的解法是1出发形成一个环，n出发也形成一个环（此时1和n的出入度都为1，其他点的出入度都为0或都为1）
5. 结：以1和n出发各跑一遍dij，找各自最小环，比较1~n的最短路和各自最小环的和，谁最小即可
6.
```
int n;
int c[MN][MN];
typedef pair<int,int> edge;        //边权，终点
priority_queue<edge, vector<edge>, greater<edge> >q;  //小根堆
int dst[MN],vst[MN];
```
7.
```
void dj(int src){  //求src源的最短路，路径存在dst里
    ms(vst,0);
    //vst[src]=1;
    for__(i,1,n)
        dst[i]=c[src][i];
    dst[src]=0x3f3f3f3f;       //自环初值无穷大
    while(q.size())
        q.pop();
    for__(i,1,n)
        q.push(make_pair(dst[i],i));
    while(q.size()){
        int x=q.top().second;
        q.pop();
        if(vst[x])            //注意x只做一次起点，需要判断vst
            continue;
        vst[x]=1;
        for__(y,1,n)
            if(dst[y]>dst[x]+c[x][y])  //注意y可以反复入堆，不用判断vst
                dst[y]=dst[x]+c[x][y],
                q.push(make_pair(dst[y],y));}}
```
8.
```
dj(1);
int ans= dst[n];            //初值1~n最短路
int cycle1= dst[1];         //1所在回路（有的话）
dj(n);
int cycleN= dst[n];
printf("%d",min(ans,cycle1+cycleN));
```

五. 星间飞行（USSTD3G，西安邀请赛原题）

1. 题意：每花c块钱能让可通行最大边权+=d，可累积经过边数+=e，求从1到n最少需花多少钱
2. 因为钱花越多能走的路越多，因此答案有单调性，二分遍历花几次钱，跑spfa算边数即可
3.
```
bool solve(ll DD,ll EE){    //跑得动边权EE的边，边总数要求<DD，能否到达
    ms(dst,0x3f);
    ms(inq,0);
    queue<int>q;
    q.push(1); inq[1]=1;
    dst[1]=0;
```

```
        while(q.size()){
            int x=q.front(); q.pop(); inq[x]=0;
            for(int i= fst[x]; i; i= nxt[i]){
                if(val[i]>DD) continue;
                int y=to[i];
                if(dst[y]>dst[x]+1){
                    dst[y]=dst[x]+1;
                    if(!inq[y]) inq[y]=1, q.push(y);}}
            if(dst[n]<=EE) return 1;}
        if(dst[n]>EE) return 0;}
```

4.
```
int l = 0, r = 10000;
while(l<r){
    int mid = l+r >>1;
    if(solve((ll)mid*d, (ll)mid*e))
        r = mid, yes = 1;
    else
        l = mid+1;
}
if(!yes) puts("-1");
else printf("%d",l*c);
```

5. 也可类似LCA的倍增

```
rof__(i,15,0)       //从2^15往2^0拆解，类似倍增求LCA
        if(!solve(D+(1<<i)*d,E+(1<<i)*e))
            D+= (1<<i) *d,
            E+= (1<<i) *e;
        else
            yes = 1;
    if(!yes) puts("-1");
    else printf("%lld",(D/d+1)*c);
```